May 19, 2017

Abstract

1 DeepIterator

Wir nennen den Iterator DeepIterator, da er mehrere hierachieebenen (deep) verbindet. Der DeepIterator wird benutzt um über verschachtelte hierachiche Datencontainer zu iterieren. Das einfachste Beispiel für eine Hierachieche Datenstruktur ist

```
1 std::vector< std::vector< int > > interleavedVector;
```

Der Deepiterator iteriert über alle int innerhalb des doppelt verschachtelten Vektors. Der Iterator benutzt den Trait *IsIndexable* um zu entscheiden ob eine Datenstruktur Listen oder Array ähnlich aufgebaut ist. Der Deepiterator wird mit mehreren Templateparametern konfiguriert:

```
1 template<typename TContainer,
2          typename TAccessor,
3          typename TNavigator,
4          typename TCollective,
5          typename TRuntimeVariables,
6          typename TChild>
7 struct DeepIterator;
```

Der erste Parameter *TContainer* gibt den Datentyp des Containers an, über dessen Componenten iteriert werden soll. TContainer muss mehrere Anforderungen erfüllen: 1. Das Trait *IsIndexable* (siehe Abschnitt 3.1) muss eine Ausprägung für den Typ TContainer haben; 2. Der Trait *ComponentType* (siehe Abschnitt 3.2) muss für TContainer ausgeprägt sein und; 3. Die Funktion *NeedRuntimeSize¡TContainer¿* muss geschrieben werden (siehe Abschnitt 3.3).

1.1 Accessor

Der Accessor beschreibt wie ein Element aus dem Container “geholt” wird. Für indexierbare Datencontainer existiert eine Überladung (siehe Abschnitt 3.1). Für andere Datenstrukturen muss eine eigener Accessor geschrieben werden. Der Accessor muss eine Funktion get besitzen:

```
1  static TComponent* Accessor::get(TContainer* con, TComponent*, TIndex&, const
```

1.2 Navigator

Der Navigator wird benutzt um das nachfolge Element zu bestimmen. Zusätzlich muss er noch eine Funktion bereitstellen um das erste Element zu bestimmen. Die Funktionen sind

```
1  static void first(TContainer* in, TContainer* out , TComponent* out, TIndex& c
2  static void next(TContainer*, TComponent*, TIndex&, RuntimeVarialbes);
```

Um indexierbare Funktionen bereitzustellen, benutzen wir den TIndex Parameter.
Wir haben eine Standardimplementierung des Navigator. Diese wird innerhalb der View (Abschnitt **??**) benutzt.

```
1 template<typename TContainer,
2           Direction TDirection>
3 struct Navigator;
```

Die *Direction* ist ein Parmeter um die Iterationsrichtung auf 1D Datenstruktur zu bestimmen. Forward startet beim ersten Element und endet beim letzten. Backward startet beim letzten Element und endet beim ersten.

1.3 Collective

1.4 RuntimeVariables

1.5 Child

2 View

sec-View

3 Traits

3.1 IsIndexable

Das Trait IsIndexable gibt an, ob ein Datencontainer eine Array-ähnliche Struktur hat. Die Vorraussetzung ist, das der Operator [] überladen ist. Ein Beispiel: t[i] gibt das i-te Element des Datencontainers t zurück.

3.2 ComponentType

Der *ComponentType¡T¿* Trait gibt den Datentyp der Compontenten von T an.

```
1  typedef std::vector<int> TIntVector;
```

3.3 NeedRuntimeSize